Script language as introduced in SynthFont2, version 2.2.2.0

Version 1.0 2019-04-11 / Kenneth Rundt

The script language is base on a simplified version of standard Pascal.

Let's take an example: We want a script that can be used to change the velocity of notes, all notes, to a certain value.

Have a look at the file "DemoScript1.pas" in the default SynthFont2 documents folder ("<My Documents>\SynthFont). The file contains these lines (and some more):

```
unit DemoScript1;
interface
procedure Change_note_velocity;
implementation

procedure Change_note_velocity;
var
 vel,ne: integer;
 md: RMidiData;
begin
  ne:= md.GetFirstEvent(0, 0);
  if ne=0 then exit;
  Debug ('Track=', md.Track, ', num items=', ne);
  vel:= AskInteger ('New velocity', 33);
  repeat
    if md.IsNote then md.Velocity:= vel;
  until md.GetNextEvent=0;
end;
…
<more procedures …>
…

end. //signals end of file
```

The section

```
interface
procedure Change_note_velocity;
implementation
```

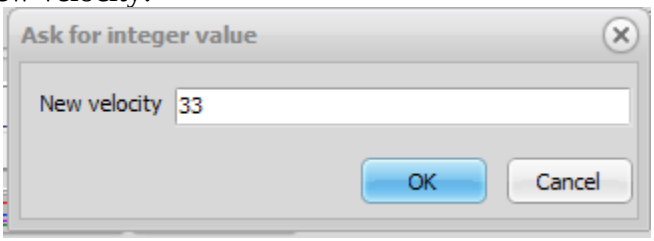
tells SynthFont2 that the file contains [at least] one procedure that can be called: "Change_note_velocity". When SynthFont2 has found this file in the default SynthFont2 folder it will read this section and make a note of all procedures listed here. These are then found in a menu item. More about this later.

After the key word "implementation" comes the implementation of all procedures. In this example we have only one (but there are more in DemoScript1 for you to study). The procedure starts by listing all the variables required by the procedure in a standard Pascal manner. In this case we will use two integers (whole numbers) called "vel" and "ne" plus a record called "md" of type "RMidiData". This is an internal record type which becomes the interface to the MIDI data.

The first executable line is "`ne:= md.GetFirstEvent(0, 0);`", making use of the record "md". This line asks for the number of MIDI events and to start the iteration process through the events at a certain event number. There are two parameters: the first is the track number. Internally tracks are numbered 1 to MaxNumTracks which is equal to the number of tracks in the MIDI file. The case of "0" means "pick the currently selected track". The second parameter is the first event to start with, normally just "0". MIDI events are numbered 0 to NumEvents-1. NumEvents is the number of MIDI events in the track. Note that as the first event is "0", the high limit is not NumEvents but one less: NumEvents-1.

The function `md.GetFirstEvent` returns the number of events starting from the second parameter ("0" in this case). Hence it is always good have this test "`if ne=0 then exit;`" on the next line. In this case, when iteration starts at "0" the value for "`ne`" is equal to NumEvents.

The line "`vel:= AskInteger ('New velocity', 33);`" is used to display a dialog box where you can enter a value for the new velocity:



Then comes the process of going through all MIDI events one-by-one. This happens normally within a "repeat until" statement group. At first we check to see if the current MIDI event is a Note event "`if md.IsNote …`". If so, we can set the velocity to the new value. In fact, it is not necessary to check if the event is a note event as Velocity can only be set to a value (`md.Velocity:= vel;`) for note events. The "`until md.GetNextEvent=0;`" retrieves the next event, or returns "false" (=0) if there are no more. Hence this line could also read "`until md.GetNextEvent=false;`" or "`until not md.GetNextEvent;`" All three alternatives are valid statements. The whole procedure code ends with "end;" in standard Pascal fashion.

Please remember that in Pascal everything is insensitive to case, hence "false" is the same as "False" or "FALSE".

The record "`RmidiData`" contains a number of variables:

`Track` – integer value equal to the current track, read only
`GetFirstEvent` – function described above
`GetNextEvent` – function described above
`AbsTickTime` – integer value equal to the absolute position of the current event in number of MIDI ticks, read/write. You can thus move events around by setting this value.
`Bar` – integer value equal to the Bar position of the current event, read only
`Beat` – integer value equal to the Beat position of the current event, read only
`Ticks` – integer value equal to the Ticks position of the current event, in the current Bar:Beat, read only
`EType` – integer value equal to the MIDI type of the current event, read only (see more below), instead of inerrogating this value it is recommended to use one of the following tests:
`IsNote` – returns "true" for a note event
`IsAfterTouch` – returns "true" for an AfterTouch event

`IsController` – returns ”true” for a Controller event
`IsMidiProg` – returns ”true” for a MIDI Program event
`IsChnlPress` – returns ”true” for a Channel Pressure event
`IsPitchWheel` – returns ”true” for a Pitch Wheel event

`Note` – integer value equal to the Note value of the current event, read/write
`CType` – integer value eqaul to the number of the Controller event, instead of tesing this value you can use one of the following tests: `IsCBankCoarse, IsCModWheel, IsCBreath, IsCFootPedal, IsCPortamentoTime, IsCVolume, IsCBalance, IsCPan, IsCExpression, IsCBankFine, IsCHoldPedal, IsCPortamento, IsCSustPedal, IsCSoftPedal, IsCLegatoPedal, IsCHoldPedal2, IsCSoundVar, IsCSoundTimbre, IsCReleaseTime, IsCAttackTime, IsCBrightness, IsCEffectsLvl, IsCTremoloLvl, IsCChorusLvl, IsCCelesteLvl`. These tests may return true only if `IsController` also returns true.
`Velocity` – integer value equal to the Note Velocity value of the current event, read/write
`Cvalue` – is the same value as Velocity, read/write
`Channel` – integer value equal to the MIDI Channel of the current event, read only (range 0 through 15)
`Length` – integer value equal to the Note Length value of the current event, read/write (valid for notes only)
`DeleteEvent` – function you can use to delete the current event.

SynthFont2 defines a few constants internally

These integer constants represent valid values for the Etype integer in RmidiData record:
TNOTE, TAFTERTOUCH, TCONTROLLER, TMIDIPROGRAM, TCHNLPRESS, TPITCHWHEEL, TMETAEVENT.

For a TCONTROLLER event (`IsCtrl` equal to true), the `Ctype` defines the type of the controller. These integer values are defined (but you can use the corresponding IsXXX instead as above):
`CBankCoarse, CModWheel, CBreath, CFootPedal, CPortamentoTime, CVolume, CBalance, CPan, CExpression, CBankFine, CHoldPedal, CPortamento, CSustPedal, CSoftPedal, CLegatoPedal, CHoldPedal2, CSoundVar, CSoundTimbre, CReleaseTime, CAttackTime, CBrightness, CEffectsLvl, CTremoloLvl, CChorusLvl, CcelesteLvl.`

To set the Pan to the center for a track you can use the constant `PanCenter` for `CValue`.

Finally, this constant is defined:
`MyDocuments` – returns the path to your Documents folder, including a trailing ”\”.

SynthFont2 defines a few records internally

One record has already been mentioned: `RMidiData`.

Another useful record is **`RAudioData`**. This record contains the following items:
`Format` – acceptable values are defined below
`BitRate` – an integer, some constants to be used are define below

SkipConverted - a boolean value, default is “false”, set to “true” to skip converted files (based on file name)
SaveLyricsInMP3 - a boolean value for use with Karaoke files, default is “false”
NoVoiceLimit - a boolean value, the default is “true”
NormalizeAudio - a boolean value, the default is “false”
Folder – a string containing the path to a folder for the audio files
FileNameAdd – a string containing a text item to add to the file name at the end, before the extension
TagArtist, TagAlbum, TagTrack, TagGenre, TagProducer, TagCopyright – strings for meta tags
Execute – call this function to render the MIDI file currently open. NOTE: you can open a MIDI file in two ways, either use OpenMidiFile or ReadMidiFile (see below).

Constants to be used for Format: These two are always available: fWAV, fWMA. These are available ONLY of the required library files (“DLL” files) are found: fMP3, fOGG, fFLAC, fAPE and fMP4.

Constants to be used for BitRate: fBRVarLow (variable bitrate, low quality), fBRVarStd, fBRVarMed, fBRVarHigh, fBRVarBest. The following integer value can be used for a constant bitrate: fBR64, fBR96, fBR128, fBR192, fBR256, fBR320. Any other value within the acceptabel range can be used.

SynthFont2 defines a few functions internally

OpenMidiFile
Use this function to open a MIDI file into the program for full access also outside of the script editor. The function takes one parameter, the file name, and returns 0 if succeeded or else an error code.

ReadMidiFile
Use this function to open a MIDI file temporarily for access only from within the running script. Faster than OpenMidiFile. The function takes one parameter, the file name, and returns 0 if succeeded or else an error code.

GetNumTracks
This function returns the number of MIDI tracks in the file open. It does NOT include any potential track layers.

SaveMidiFile
When done changing everything you want to change, in all tracks you want to change – save the file.

WriteAudioFile
An alternative to using the record RAudioData defined above.
This function can take up to five parameters:
WriteAudioFile; - simply create a WAV file in the same folder using the name of the MIDI file.
WriteAudioFile (fWAV); - same as above, instead of fWAV you can use fMP3 .. fMP4.
WriteAudioFile (fMP3); - create a MP3 file with bit rate 128
WriteAudioFile (fMP3, fBR192); - create a MP3 file with bit rate 192
WriteAudioFile (fMP3, 192); - create a MP3 file with bit rate 192
WriteAudioFile (fMP3, fBRVarHigh); - create a MP3 file with variable bit rate, high quality
WriteAudioFile (fMP3, fBRVarHigh, True); - as above, but Normalize the volume levels

`WriteAudioFile (fMP3, fBRVarHigh, False, MyDocuments+'MP3Audio');` - create a MP3 file with variable bit rate, high quality, no normalization, and put the file into a folder called MP3Audio found in "My Documents".
`WriteAudioFile (fMP3, fBR192, True, '', '-192);` - create a normalized MP3 file with bit rate 192, into the default folder but add the text '-192' to the file name before the extension '.mp3'.

`Debug`
Takes a variable number of parameters and creates text written into the information box of the script editor dialog.

`StopNow`
When running long iterations in a script you may want to check this function regularly within the iteration loop. It returns True if you have pressed the button "Stop now" in the script editor dialog. Then you can exit the script to interrupt the execution.

`AskInteger`
Displays a dialog box where you can enter an integer value. Takes as parameters a string for a label and a starting value.

`AskFloat`
Displays a dialog box where you can enter a floating point value. Takes as parameters a string for a label and a starting value.

`BrowseForFolder`
Search for a folder on your computer. Takes a string as a parameter for the folder where to start. If the string is empty ('') the last used folder will be the starting point. Returns a string with the selected folder.

`GetFilesInFolder`
Takes two parameters: the folder to search and a filter value, like '*.mid' for files with the extension '.mid'. Returns an array of file names.

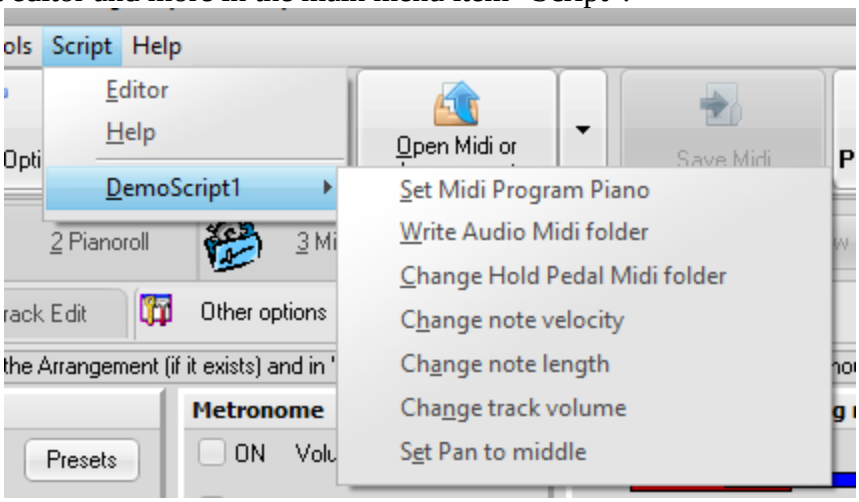
`MyDocuments`
Returns the path to the Documents folder. All script files must be here in the "SynthFont" folder.

`ShowEditor`
Use in connection with a length procedure to make sure the script editor is visible to give you the option to hit "Stop now".

`HideEditor`
Simply hide the script editor.

The script editor

You find the script editor and more in the main menu item “Script”.



Choose ”Editor” to open the actual editor window. ”Help” will open this document.

You will find the demo script file DemoScript1 in the “<My Documents>\SynthFont” folder. The file contains seven procedures to study and to use, for example, the “Change note velocity” procedure described in the introduction. When SynthFont2 finds a name of a procedure in the interface section of a script file, it creates a menu item as a shortcut, as shown above. The caption of the menu item is derived from the procedure name by replacing undescores with spaces: “Change_note_velocity” becomes “Change note velocity”.